Part2 Problems 8 to 14

SPPU DBMS PRACTICALS - SET II
Part 2: Problem Statements 8 - 14
==========

Problem Statement 4 — Aggregation & Indexing (MongoDB)

Collection: Student_Data(Student_ID, Student_Name, Department, Marks)

Commands & Inserts:
use College;
db.createCollection("Student_Data");
db.Student_Data.insertMany([
{ Student_ID: 1, Student_Name: "Vivek", Department:"CS", Marks: 78 },
{ Student_ID: 2, Student_Name: "Priya", Department:"IT", Marks: 85 },
{ Student_ID: 3, Student_Name: "Rohan", Department:"CS", Marks: 92 },
{ Student_ID: 4, Student_Name: "Sneha", Department:"EXTC", Marks: 71 },
{ Student_ID: 5, Student_Name: "Kiran", Department:"IT", Marks: 66 },
{ Student_ID: 6, Student_Name: "Amit", Department:"CS", Marks: 59 },
{ Student_ID: 7, Student_Name: "Neha", Department:"EXTC", Marks: 95 }
]);

Queries:
1. db.Student_Data.aggregate([{ $group: { _id: "$Department", AvgMarks: { $avg: "$Marks" }, Students: { $push: "$Student_Name" } } }]);
2. db.Student_Data.aggregate([{ $group: { _id: "$Department", Count: { $sum: 1 } } }]);
3. db.Student_Data.aggregate([{ $sort: { Marks: -1 } }, { $group: { _id: "$Department", TopStudent: { $first: "$Student_Name" }, TopMarks: { $first: "$Marks" } } }, { $sort: { TopMarks: -1 } }]);
4. db.Student_Data.createIndex({ Student_ID: 1 });
5. db.Student_Data.createIndex({ Student_Name: 1, Department: 1 });
6. db.Student_Data.dropIndex({ Student_ID: 1 });
7. db.Student_Data.dropIndex({ Student_Name: 1, Department: 1 });

Problem Statement 5 — DML using MySQL

Tables: Customer(CustID, Name, Cust_Address, Phone_no, Age)
Branch(BranchID, Branch_Name, Address)
Account(Account_no, BranchID, CustID, date_open, Account_type, Balance)

SQL (create tables omitted - same as earlier)

Tasks:
1. ALTER TABLE Customer ADD Email_Address VARCHAR(100);
2. ALTER TABLE Customer CHANGE Email_Address Email_ID VARCHAR(100);
3. SELECT c.*, a.Balance, a.Account_no FROM Customer c JOIN Account a ON c.CustID = a.CustID WHERE a.Balance = (SELECT MAX(Balance) FROM Account);
4. SELECT c.*, a.Balance, a.Account_no FROM Customer c JOIN Account a ON c.CustID = a.CustID WHERE a.Account_type = 'Saving Account' AND a.Balance = (SELECT MIN(Balance) FROM Account WHERE Account_type = 'Saving Account');
5. SELECT * FROM Customer WHERE Cust_Address = 'Pune' AND Age > 35;
6. SELECT CustID, Name, Age FROM Customer ORDER BY Age ASC;
7. SELECT a.Account_type, c.Name, a.BranchID FROM Customer c JOIN Account a ON c.CustID = a.CustID GROUP BY a.Account_type, c.Name, a.BranchID;

Problem Statement 6 — Procedures / Functions (PL/SQL)

Schema: Employee(emp_id, dept_id, emp_name, DoJ, salary, commission, job_title)

1. Stored Procedure to record commission:
CREATE OR REPLACE PROCEDURE calc_commission AS
BEGIN

```
FOR rec IN (SELECT emp_id, salary, DoJ FROM Employee) LOOP
DECLARE
comm NUMBER(10,2);
yrs NUMBER := FLOOR(MONTHS_BETWEEN(SYSDATE, rec.DoJ)/12);
BEGIN
IF rec.salary > 10000 THEN
comm := rec.salary * 0.004;
ELSIF rec.salary < 10000 AND yrs > 10 THEN
comm := rec.salary * 0.0035;
ELSIF rec.salary < 3000 THEN
comm := rec.salary * 0.0025;
ELSE
comm := rec.salary * 0.0015;
END IF;
UPDATE Employee SET commission = comm WHERE emp_id = rec.emp_id;
END;
END LOOP;
COMMIT;
END;
/
```

2. Function get_manager_name(p_dept_id NUMBER) RETURN VARCHAR2

```
CREATE OR REPLACE FUNCTION get_manager_name(p_dept_id NUMBER) RETURN VARCHAR2 IS
v_manager VARCHAR2(100);
BEGIN
SELECT Manager_name INTO v_manager FROM Departments WHERE Department_id = p_dept_id;
RETURN v_manager;
EXCEPTION
WHEN NO_DATA_FOUND THEN
RETURN 'No Manager';
END;
/
```

Problem Statement 7 — Map Reduce (MongoDB)

Collection: Book(Title, Author_name, Borrowed_status, Price)

```
db.createCollection("Book");
db.Book.insertMany([
{ Title:"Book A", Author_name:"Author1", Borrowed_status:true, Price:350 },
{ Title:"Book B", Author_name:"Author1", Borrowed_status:false, Price:250 },
{ Title:"Book C", Author_name:"Author2", Borrowed_status:true, Price:400 },
{ Title:"Book D", Author_name:"Author3", Borrowed_status:true, Price:150 }
]);
```

1. map-reduce author wise list of books:

```
var map1 = function() { emit(this.Author_name, this.Title); };
var reduce1 = function(key, values) { return values; };
db.Book.mapReduce(map1, reduce1, { out: "author_books" });
```

2. borrowed status true:

```
var map2 = function() { if (this.Borrowed_status === true) emit(this.Author_name, this.Title); };
var reduce2 = function(key, values) { return values; };
db.Book.mapReduce(map2, reduce2, { out: "author_borrowed_books" });
```

3. price > 300:

```
var map3 = function() { if (this.Price > 300) emit(this.Author_name, this.Title); };
var reduce3 = function(key, values) { return values; };
db.Book.mapReduce(map3, reduce3, { out: "author_pricey_books" });
```